ELSEVIER

Computational Geometry 17 (2000) 165–188

Computational Geometry
Theory and Applications

www. elsevier.nl/locate/comgeo

Drawing series parallel digraphs symmetrically ☆

Seok-Hee Hong [a,*,1], Peter Eades [a], Sang-Ho Lee [b]

[a] *Basser Department of Computer Science, University of Sydney, Sydney, Australia*
[b] *Department of Computer Science, Ewha Womans University, Korea*

Communicated by M. Attalah; received 22 May 2000; received in revised form 10 August 2000; accepted 30 August 2000

Abstract

In this paper we present algorithms for drawing series parallel digraphs with as much symmetry as possible. The first step is to compute a certain kind of automorphism, called an "upward planar automorphism" for an input series parallel digraph. The next step uses these automorphisms to construct a symmetric drawing of the graph. We present several variations of the second step, with visibility drawings, "bus-orthogonal" drawings, and polyline drawings. All algorithms run in linear time. © 2000 Elsevier Science B.V. All rights reserved.

1. Introduction

Series parallel digraphs are one of the most common types of graphs: they appear in flow diagrams, dependency charts, and in PERT networks. Algorithms for drawing series parallel digraphs have appeared in [4,6].

Symmetry is much admired in graph drawings. Winning entries for the graph drawing competitions of 1992–1998 have included many drawings which display some kind of symmetry. Algorithms for drawing graphs symmetrically have been developed for trees, outerplanar graphs, and embedded planar graphs by Manning and Atallah [12–14]. Eades and Lin [8,10] has shown that many "force directed" algorithms can be used to display symmetry. Manning [14], Bachl [3] and Chin and Yen [5] have analyzed the complexity of drawing graphs symmetrically under a variety of models.

In this paper we describe an algorithm which draws series parallel digraphs with as much symmetry as possible. Sample drawings are in Fig. 1.

☆ This research has been supported by an Australian Research Council Grant, KOSEF postdoctoral fellowship, and the SCARE project at the University of Limerick.

* Corresponding author.
E-mail addresses: shhong@cs.usyd.edu.au (S.-H. Hong), peter@cs.usyd.edu.au (P. Eades), shlee@mm.ewha.ac.kr (S.-H. Lee).

[1] This paper was partially written when the first author was visiting the University of Newcastle, the University of Limerick, and the Max Planck Institute for Informatics.

PII: S0925-7721(00)00020-1

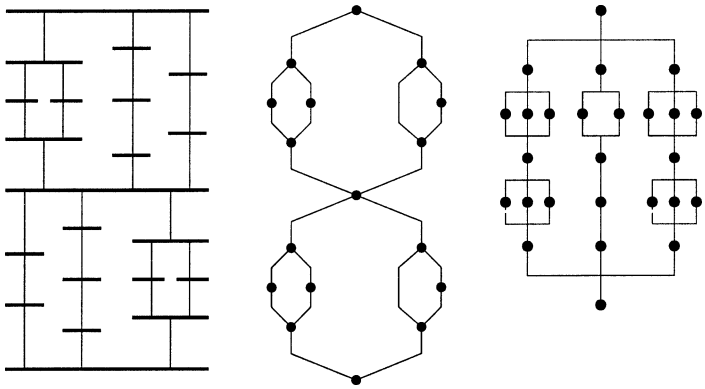

Fig. 1. Drawings output by the algorithms described in this paper.

This paper is organized as follows. In the next section, we summarize the necessary background for series parallel digraphs and symmetric drawings. The algorithm has two parts. The first part, described in Section 3, is the computation of appropriate subgroups of the automorphism group of the input graph. The second part, described in Section 4, uses these automorphisms in a few drawing algorithms. Section 5 concludes.

2. Background

2.1. Series parallel digraphs

First we review some of the fundamental notions for series parallel digraphs [2]. A digraph consisting of two vertices u and v and a single edge from u to v is a series parallel digraph with source u and sink v. If $G_1, G_2, \ldots, G_k$ are series parallel digraphs, then so are the digraphs constructed by each of the following operations:

- *series* composition: identify the sink of G_i with the source of G_{i+1}, for $1 \leqslant i < k$,
- *parallel* composition: identify all the sources of G_i for $1 \leqslant i \leqslant k$, and identify all the sinks of G_i for $1 \leqslant i \leqslant k$.

A *component* of G is one of the subgraphs $G_1, G_2, \ldots, G_k$. Series parallel digraphs may be represented as decomposition trees [17], such as in Fig. 2(a). Leaf nodes in the tree represent edges in the series parallel digraph, and internal nodes are labeled S or P to represent series or parallel compositions. The components of a series composition are sorted from source to sink; we ensure that the left-right order of the children of the corresponding node in the decomposition tree is the same. However, children of a node corresponding to a parallel composition may be ordered arbitrarily.

Because parallel composition is commutative and both series and parallel compositions are associative, there may be more than one decomposition tree for a series parallel digraph. For example, the two trees in Fig. 2 describe the same graph. Most of the literature on series parallel graphs uses a *binary* decomposition tree, with $k = 2$, such as in Fig. 2(a). However, since it is not unique, an algorithm based

[2] The class of graphs discussed in this paper are called "edge series parallel digraphs" by Valdes et al. [17].

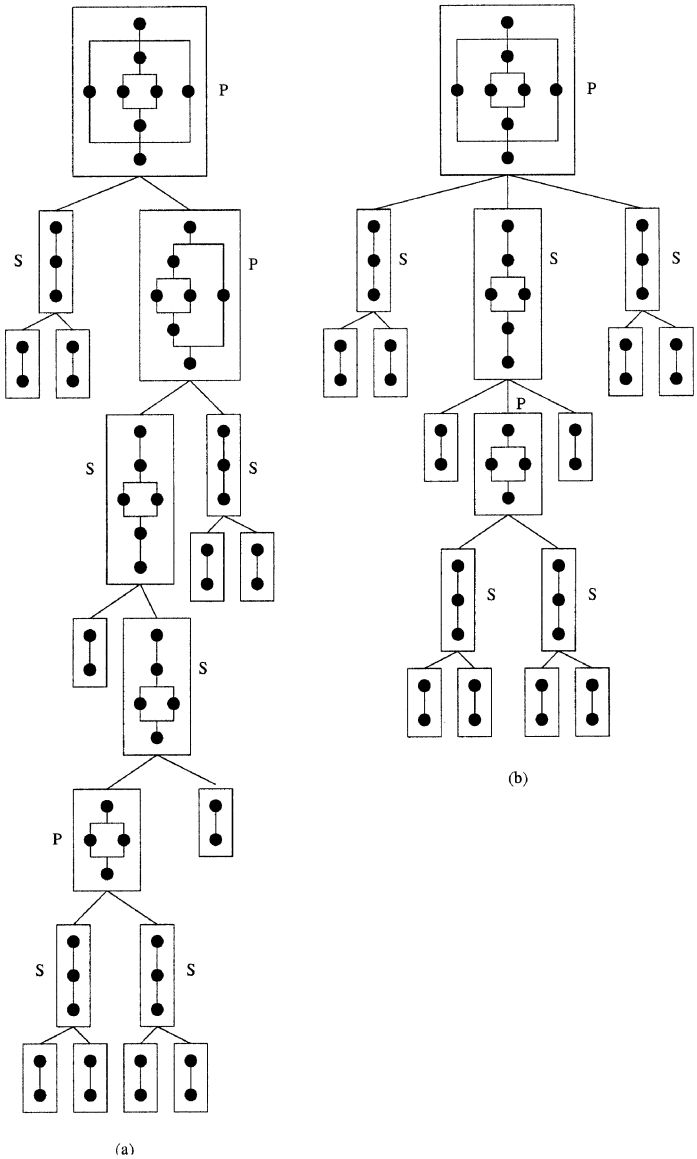

Fig. 2. Two decomposition trees of the same graph: (a) a binary decomposition tree, and (b) the canonical decomposition tree.

on the binary decomposition tree cannot fully display symmetries. To overcome this we use the *canonical decomposition tree* in which the same composition operations are placed at the same depth. Fig. 2(b) shows the canonical decomposition tree. It is easy to show that the canonical decomposition tree is unique up to the ordering of the children of P nodes.

The canonical decomposition tree can be computed in linear time using the algorithm of Valdes et al. [17,18] followed by a simple depth-first-search restructuring operation.

2.2. Upward planar drawings

This paper is concerned with *upward planar* drawings, that is, planar drawings in which each directed edge is monotonically increasing in the y direction. There is a great deal of literature regarding drawings of upward planar digraphs (see [7]); however, series parallel digraphs form a small and relatively simple subclass, and special methods apply. The best known method is the *Δ-algorithm* [4,6], which produces a straight-line grid drawing of a series parallel digraph. At best, the Δ-algorithm displays a subset of the set of possible symmetries. The methods in Section 4 below displays all possible symmetries.

2.3. Automorphisms, symmetries, and geometric automorphisms

To ensure that *all possible* symmetries are displayed, it is important to use a rigorous model for the intuitive concept of symmetry display. In this section we describe such a model, derived from those introduced by Manning [12–14] and Lin [10].

Symmetries of a graph drawing correspond to automorphisms of the graph. However, some automorphisms cannot be displayed as symmetries of any graph layout. Further, it is possible to have two automorphisms, each of which can be displayed, but for which there is no drawing which displays both. See [8,10] for examples. For these reasons, we define special kinds of automorphisms, and indicate how these relate to the symmetries of graph drawings.

Automorphisms and isomorphism partitions. An *automorphism* of an undirected graph is a permutation of the vertex set which preserves adjacency of vertices. For a directed graph $G = (V, E)$, there are two kinds of automorphism. A *direction preserving* automorphism is a permutation p of V such that $(u, v) \in E$ if and only if $(p(u), p(v)) \in E$, and a *direction reversing* automorphism is a permutation q of V such that $(u, v) \in E$ if and only if $(q(v), q(u)) \in E$. The set of all automorphisms (direction preserving and reversing) forms a group called the *automorphism group* of G.

An *isomorphism partition* of a set $\mathcal{G}$ of graphs is a partition of $\mathcal{G}$ into subsets $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m$ such that two graphs are in the same subset if and only if they are isomorphic. The sets $\mathcal{G}_i$ are called *isomorphism classes*. The partition is usually expressed by assigning an integer $code(G)$ to every graph $G \in \mathcal{G}$ such that, for each G, $G' \in \mathcal{G}$, $code(G) = code(G')$ if and only if G is isomorphic to G'.

Finding an automorphism group of a graph is *isomorphism complete*, that is, equivalent to testing whether two graphs are isomorphic. This problem is not known to have a polynomial time algorithm, nor is it known to be NP-complete. For some special classes of graphs, for example, graphs of bounded degree and planar graphs, the isomorphism problem has polynomial-time algorithms. In such cases, isomorphism partitions of sets of graphs can also be found efficiently.

In particular, there are simple linear time algorithms for testing isomorphism of trees [1]. Since series parallel digraphs have a tree-like structure, we can borrow some of the techniques for trees, in Section 3.1.

For more details on the isomorphism problem, see [2].

Symmetries and geometric automorphisms. We are interested in those automorphisms which can be represented geometrically as a symmetry of an upward planar graph drawing. The symmetries of a bounded set of points in the plane (such as a two dimensional graph drawing) form a group called the *symmetry group* of the set. The structure of symmetry groups has been studied for centuries.

A symmetry α of a drawing D of a graph G *induces* an automorphism p of G if the restriction of α to the points representing vertices of G is p. A drawing D of a graph G *displays* an automorphism p of G if there is a symmetry α of D which induces p. An automorphism p is *geometric* if there is a drawing D of G which displays p. Further, D *displays* a subgroup P of the automorphism group of G if D displays every element of P, and P is a *geometric automorphism group* if there is a drawing D which displays every element of P.

Lin [10] gave a group-theoretic characterization of geometric automorphism groups of undirected graphs. To state this, we need some of the terminology of permutation groups; for more details see [19]. A group which contains only the identity permutation is *trivial*. The group generated by $p_1, p_2, \ldots, p_k$ is denoted by $\langle p_1, p_2, \ldots, p_k\rangle$. If a permutation p acting on a set V has a *fixed element* $v \in V$, that is, $p(v) = v$, then p induces a permutation p_v on $V - \{v\}$. A permutation group P *is semiregular* if each non-identity permutation in P does not have a fixed element. A non-identity permutation p on V is a *rotational* permutation if either $\langle p\rangle$ or $\langle p_v\rangle$ (for some $v \in V$) is semiregular. Note that a rotational permutation has at most one fixed element.

A permutation p has *order* k if k is the smallest positive integer such that p^k is the identity.

A permutation p *fixes* the subset U of V if $p(u) \in U$ for each $u \in U$. Note that an element of a fixed subset is not necessarily a fixed element; for this reason we sometimes say that p fixes U *setwise*. If p is an automorphism of a graph G and H is a subgraph of G, then p *fixes* H if p fixes the vertex set of H setwise.

Theorem 1 (Lin [10]). *A subgroup P of the automorphism group of a graph $G = (V, E)$ is geometric if and only if the permutation group on V defined by P is one of the following types*:

1. $P = \langle q\rangle$ *where q has order 2*; *or*
2. $P = \langle p\rangle$ *where p is a rotational permutation*; *or*
3. $P = \langle p, q\rangle$ *such that*:
   (a) *p is a rotational permutation and q has order 2, and*
   (b) $\langle p\rangle \cap \langle q\rangle$ *is trivial, and*
   (c) $qp = p^{-1}q$.

Note that some automorphisms of order 2 can be displayed as rotations by 180°, some can be displayed as reflections in a line (or axis), and some can be displayed as either. We say an automorphism is *axial* if it can be displayed as a reflection in a line.

For general undirected graphs, the problem of finding a geometric automorphism of a graph is NP-hard [11,14]. This means that it may be strictly harder than the problem of finding the automorphisms of graphs in general (which is merely isomorphism hard [15]). For planar undirected graphs, it can be shown that planar geometric automorphisms can be found in polynomial time [9]; for some special classes of planar graphs (trees and outerplanar graphs) it is linear time [12,13].

Upward planar automorphisms. The concept of "geometric automorphism group" can be easily refined to upward planar drawings of digraphs. A geometric automorphism group P of a digraph G is an *upward planar automorphism group* if there is an upward planar drawing of G which displays each element of P. Theorem 1 gives necessary but not sufficient conditions for a group to be an upward planar automorphism group. For example, consider the digraph in Fig. 3: as an undirected graph it has a

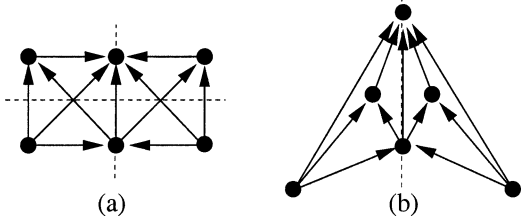

Fig. 3. Two drawings of an upward planar digraph.

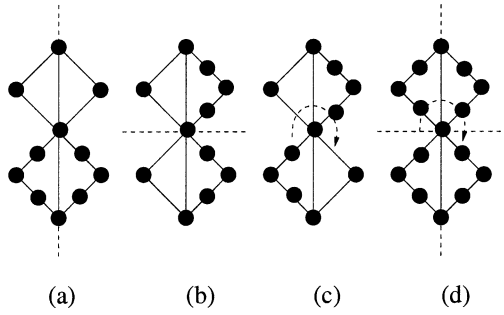

Fig. 4. (a) Vertical automorphism, (b) horizontal automorphism, (c) rotational automorphism, (d) group of size 4.

geometric automorphism group of size 4, as displayed in Fig. 3(a). However, the largest upward planar automorphism group has size 2, as shown in Fig. 3(b).

In the next section we give an algorithm for computing the maximum size upward planar automorphism group of a series parallel digraph. In Section 4 we give a variety of drawing algorithms that can be used to display such a group.

3. Upward planar automorphisms of series parallel digraphs

In this section we present an algorithm which finds upward planar automorphisms for series parallel digraphs. The automorphism group obtained in this way is used in Section 4 to draw the graph symmetrically.

In fact the upward planar automorphisms of a series parallel digraph are quite simple. Fig. 4 shows four symmetric drawings. In this figure and many that follow, edges are directed upward, and arrowheads are omitted.

- Fig. 4(a) displays a *vertical* automorphism, that is, an automorphism induced by a reflection in the y axis. This is a direction-preserving automorphism of order 2, and it fixes the source and fixes the sink.
- Fig. 4(b) displays a *horizontal* automorphism, that is, an automorphism induced by a reflection in the x axis. This is a direction-reversing automorphism of order 2; it swaps the source and the sink.
- Fig. 4(c) displays a *rotational* automorphism of order 2. This is an automorphism induced by a rotation by 180°. It reverses directions, and swaps the source and the sink.
- Fig. 4(d) displays an upward planar automorphism group of size 4, containing one of each of the types vertical, horizontal and rotational.

In fact, Fig. 4 covers all the nontrivial possibilities for an upward planar automorphism group of a series parallel digraph.

Lemma 1. *An upward planar automorphism group of a series parallel digraph is either*

- *trivial, or*
- $\{1, p\}$ *where* p *is either vertical, horizontal, or a rotation of order 2, or*
- $\{1, p, q, r\}$*, where the non-identity elements are one of each of the types vertical, horizontal, or rotation of order 2.*

Proof. Every automorphism of a series parallel digraph fixes the set consisting of the source and the sink. This means that every automorphism has order 2. Using Theorem 1, one can deduce the lemma. □

To find all upward planar automorphisms, we search for each of the types above. Roughly speaking, the method proceeds as follows.

1. Construct the canonical decomposition tree. This can be done using the method of Valdes [17,18] followed by a depth-first search.
2. Check for the existence of each of the upward planar automorphisms mentioned in Lemma 1.
   (a) Check for vertical automorphism.
   (b) Check for horizontal automorphism.
   (c) Check for rotational automorphism.
3. Compute the maximum upward planar automorphism group.

Step 1 is quite simple, and we will not discuss it further. Steps 2(a), (b) and (c) are described in Sections 3.1, 3.2 and 3.3, respectively. Step 3 is described in Section 3.4.

Two remarks are in order before we proceed.

- Note that Lemma 1 does not hold for undirected series parallel graphs. For example, a cycle of size three is a series parallel graph and has a geometric automorphism group of size 6.
- Step 3 is not simply writing out the results of step 2. For example, the existence of a horizontal automorphism plus the existence of a rotational automorphism is not sufficient to ensure the existence of an upward planar automorphism group of size 4. In fact, a single automorphism can be both horizontal and rotational, as shown by Figs. 4(b) and (c). Note that there is no drawing of this graph which has both horizontal and rotational symmetry.

3.1. Vertical automorphisms

Vertical automorphism is the easiest to detect of the three types mentioned in Lemma 1. We need two substeps.

- First, we label the canonical decomposition tree. The labeling is canonical, in the sense that isomorphic components have equal labels.
- Next, we use a recursive method to find the vertical automorphisms.

Vertical labeling. We label each node of the canonical decomposition tree in such a way that nodes corresponding to isomorphic components receive the same label. The method is essentially an adaptation of a tree isomorphism algorithm [1].

The *depth* of a node v in the canonical decomposition tree is the distance of v from the root. Automorphisms of series parallel graphs preserve the depths, in a sense defined by the following simple lemma.

Lemma 2. *Suppose that α is an automorphism of a series parallel digraph G with canonical decomposition tree T. If H is a component of G corresponding to the node u of T, then $\alpha(H)$ is a component whose corresponding node in the canonical decomposition tree has the same depth as u.*

Proof. This can be proved by induction on the canonical decomposition tree, from the nodes at maximum depth to the root. □

The labeling algorithm first computes the depth of each node. Then it assigns labels to the nodes, starting with vertices at the maximum depth and working up towards the root. Each node u receives two labels: an integer $code_V(u)$, and a sequence $tuple_V(u)$ of integers. The values of $code_V(u)$ define an isomorphism partition of the components with the same depth as u. The sequence $tuple_V(u)$ defines an isomorphism partition of the children of u, but it is sorted in a special way. For a series node, it is sorted by the left-right order of the nodes (that is, by the upward order of the components). For a parallel node, the sorting is more complex.

`vertical_labeling` (T)
1. Compute the depth of each node.
2. Initialize the tuples for each leaf u of T: $tuple_V(u) = (0)$.
3. Repeat for each depth i, from the maximum depth to the root:
   (a) For each internal node u of T at depth i, $tuple_V(u) = (code_V(v_1), code_V(v_2), \ldots, code_V(v_k))$, where the children of u are $v_1, v_2, \ldots, v_k$, from left to right.
   (b) If u is a P node, then sort $tuple_V(u)$.
   (c) Let Q be the list of tuples for the nodes of T at depth i. Sort Q lexicographically.
   (d) For each node u of T at depth i, compute $code_V(u)$ as follows. Assign the integer 1 to those nodes of T at depth i represented by the first distinct tuple of the sorted list Q, assign the integer 2 to the nodes represented by the second distinct tuple, and so on.

Lemma 3. *The time and space complexities of algorithm* `vertical_labeling` *are linear.*

Proof. Steps 1 and 2 are clearly linear. Consider iteration i of the main loop (step 3). Let k denote the total number of children of nodes of depth i. Steps 3(a) and (d) are clearly linear in k. Steps 3(b) and (c) can be executed in $O(k)$ time and space using a bin sorting approach. The lemma follows by summing over all iterations of the main loop. □

Fig. 5 shows an example of the labeling process for a canonical decomposition tree.

Effectively, the labeling computes an isomorphism partition of the nodes at each depth, as stated in the following lemma.

Lemma 4. *Suppose that u and v are nodes in the canonical decomposition tree T of a series parallel digraph G, and that u and v have the same depth. Then the component represented by u is isomorphic to the component represented by v if and only if $code_V(u) = code_V(v)$.*

Proof. This may be proved by a simple inductive argument, following the labeling algorithm above. □

Finding vertical automorphisms. Next we give an algorithm for detecting whether a series parallel digraph has a vertical automorphism. The algorithm depends crucially on the components which are fixed (setwise) by a vertical automorphism. These are classified in the following lemma.

Lemma 5. *Suppose that G is a series parallel digraph, where the children of the root in the canonical decomposition tree of G represent the components $G_1, G_2, \ldots, G_k$. Suppose that α is a vertical automorphism of G.*

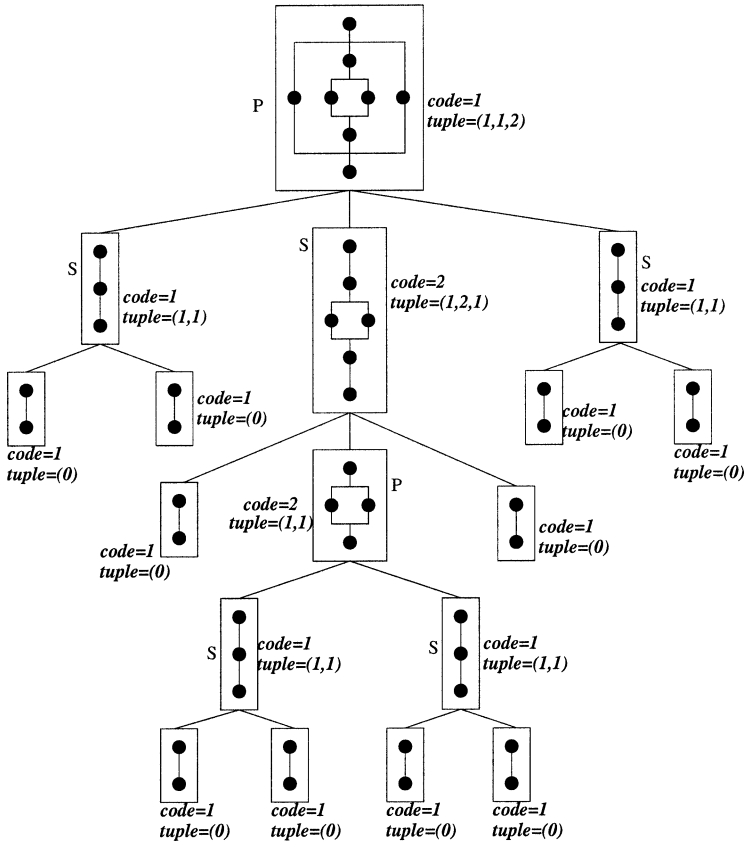

Fig. 5. Labels on a canonical decomposition tree.

1. *If G is a series composition, then α fixes each one of $G_1, G_2, \ldots, G_k$.*
2. *If G is a parallel composition, then α fixes at most one of $G_1, G_2, \ldots, G_k$.*

Proof. We deal with the series composition first. Suppose that the components $G_1, G_2, \ldots, G_k$ of a series composition are in order from source to sink. Note that G_1 is a leaf of the block-cutvertex tree of G, and so α must map G_1 to a leaf of the block-cutvertex tree. However, G_1 and G_k are the only leaves of the block-cutvertex tree. Since α is a vertical automorphism, it fixes the source of G, which is in G_1 and not in G_k. Hence α cannot map G_1 to G_k, and must map G_1 to itself. Thus α fixes G_1, and fixes the source and sink of G_1. The sink of G_1 is the source of G_2, and we can continue the argument to show that α fixes each of $G_1, G_2, \ldots, G_k$.

Now suppose that G is a parallel composition of $G_1, G_2, \ldots, G_k$. Suppose that α fixes two components G_1 and G_2, and that D is a drawing of G which displays α. Suppose that s and t are the (common) source and sink of G_1 and G_2. Since the decomposition tree is canonical, G_i is either a single edge or a series composition, for $i = 1, 2$.

Assume that both G_1 and G_2 are series compositions. This means that each contains a cutvertex. Since G_1 is connected, there is a path p_1 in G_1 from s to t. Now $\alpha(p_1)$ is also a path from s to t in G_1 (since α fixes G_1). The paths p_1 and $\alpha(p_1)$ must intersect at least once (because G_1 has a cutvertex); each vertex

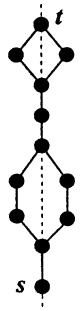

Fig. 6. Paths used in the proof of Lemma 5.

at which they intersect must lie on the straight-line segment between s and t. Where they are disjoint they must be axially arranged about the y axis, as in Fig. 6. Similarly, G_2 contains a path p_2 from s to t, and p_2 with $\alpha(p_2)$ forms a similar picture. This clearly contradicts planarity.

The case where at one of G_1 and G_2 is a single edge is similar; this completes the proof. □

From Lemma 5 one can derive the following theorem, which forms the basis of the algorithm to follow.

Theorem 2. *Suppose that G is a series parallel digraph.*

1. *If G is a series composition of $G_1, G_2, \ldots, G_k$, then G has a vertical automorphism if and only if each one of $G_1, G_2, \ldots, G_k$ has a vertical automorphism.*
2. *Suppose that G is a parallel composition of $G_1, G_2, \ldots, G_k$. Consider the isomorphism partition of $G_1, G_2, \ldots, G_k$.*
   (a) *If there is more than one isomorphism class with an odd number of elements, then G has no vertical automorphism.*
   (b) *If all isomorphism classes have an even number of elements, then G has a vertical automorphism.*
   (c) *If one isomorphism class has an odd number of elements, then G has a vertical automorphism if and only if the component of the odd size isomorphism class has a vertical automorphism.*

Proof. First consider the series composition case. If G has a vertical automorphism, then Lemma 5 implies that each component has a vertical automorphism. Conversely, suppose that each of $G_1, G_2, \ldots, G_k$ has a vertical automorphism. This means that each G_i has a drawing which displays an automorphism which fixes the source and sink of G_i. By arranging these drawings on a vertical line, as in Fig. 7(a), we get a drawing of G which displays vertical automorphism.

Now suppose that G is a parallel composition of $G_1, G_2, \ldots, G_k$.

Note that if an isomorphism class has an odd number of elements, then a vertical automorphism must fix one of the components in this class.

If there is more than one isomorphism class with an odd number of elements, then there must be more than one fixed component. Case 2(a) immediately follows from Lemma 5.

If all isomorphism classes have an even number of elements, then we can group $G_1, G_2, \ldots, G_k$ into isomorphic pairs. For each pair G_i, G_j, we can construct drawings D_i of G_i and D_j of G_j such that D_i is a mirror image of D_j. These drawings can be transformed to a "croissant-shape" formed by two parabolas. To do this, we compute the bounding rectangle of the drawing, and linearly map each horizontal line segment through the bounding rectangle to a horizontal line segment at the same y

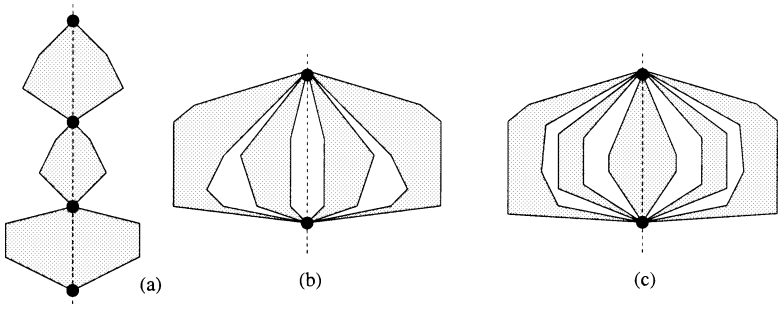

Fig. 7. Vertical arrangements.

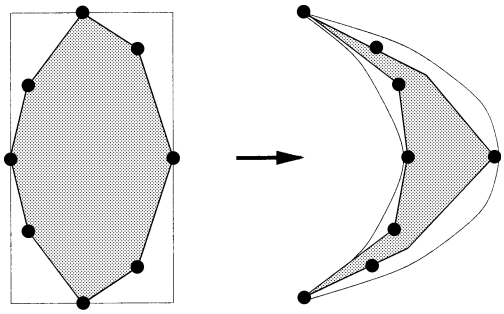

Fig. 8. The croissant transformation.

coordinate through the croissant. The mapping is one-one except at the top and bottom, where there is only one vertex. It is easy to see that it preserves planarity. The transformation is illustrated in Fig. 8.

We can construct a drawing of G which displays a vertical automorphism by applying the croissant transformation to each component, then arranging each isomorphism pair symmetrically on the opposite sides of a vertical line as in Fig. 7(b). This implies that G has a vertical automorphism.

Suppose that one isomorphism class has an odd number of elements. If G has a vertical automorphism, then this fixes a member of the odd class; indeed, it must be a vertical automorphism on that member. Conversely, suppose that the component of the odd size isomorphism class has a vertical automorphism. Then, we can construct a drawing of G which displays a vertical automorphism, by arranging each (croissant-transformed) isomorphic pair symmetrically on the opposite sides of a vertical line and placing the component on the vertical line as in Fig. 7(c). □

Suppose that r is the root node of the canonical decomposition tree T of a series parallel digraph G. The overall algorithm can be described recursively as follows.

`vertical_check`(r)
If r is an S node
then
  if `vertical_check`(v) for every child v of r
  then return(*true*)
  else return(*false*)

else if r is a P node
then
(a) Partition the children of r into classes with equal values of $code_V$ (that is, into isomorphism classes).
(b) If all the sizes of the classes are even, then return(*true*).
(c) If more than one class has odd size, then return(*false*).
(d) If only one class has odd size, then choose some v in this class and return(`vertical_check`(v)).

It is clear that algorithm `vertical_check` runs in linear time.

3.2. Horizontal automorphisms

A horizontal automorphism is direction-reversing, and detection requires a "reversing operation", before labeling. Thus we need three steps to find horizontal automorphisms:

- First, we apply the "reversing operation", which marks some S nodes as "reversed".
- The next step is a labeling step, similar to that for vertical automorphisms, but taking the reversals into account.
- Finally, we use a recursive method to find the horizontal automorphisms.

We now describe each of these steps.

The reversing operation for horizontal automorphisms. This step effectively reverses the order of the children of S nodes of half of the graph G.

Suppose that r is the root node of the canonical decomposition tree T of a series parallel digraph G. The reversal marks some S nodes as "reversed", as follows.

`horizontal_reverse`(r)
If r is a P node
then `horizontal_reverse`(v) for every child v of r
else if r is an S node
then
(a) Let $v_1, v_2, \ldots, v_k$ be the children of r, in the order of their composition, and suppose that $m = \lceil k/2 \rceil$.
(b) Mark each S node in the subtrees of the canonical decomposition tree T rooted at the nodes $v_{m+1}, \ldots, v_k$ as "reversed".
(c) If k is odd then `horizontal_reverse`(v_m).

Horizontal labeling. The labeling step for horizontal automorphisms is very similar to that for vertical automorphisms; the only difference is that the reversals indicated by the nodes marked in the reversal operation is used. We compute two labels $tuple_H(u)$ and $code_H(u)$ as follows.

`horizontal_labeling`(T)
1. Compute the depth of each node.
2. Initialize the tuples for each leaf u of T: $tuple_H(u) = (0)$.
3. Repeat for each depth i, from the nodes at maximum depth to the root:
   (a) For each internal node u of T at depth i, suppose that the children of u are $v_1, v_2, \ldots, v_k$, from left to right.

if u is marked as "reversed",
then $tuple_H(u) = (code_H(v_k), code_H(v_{k-1}), \ldots, code_H(v_1))$;
else $tuple_H(u) = (code_H(v_1), code_H(v_2), \ldots, code_H(v_k))$.

(b) If u is a P node, then sort $tuple_H(u)$.
(c) Let Q be the list of tuples for the nodes of T at depth i. Sort Q lexicographically.
(d) For each node u of T at depth i, compute $code_H(u)$ as follows. Assign the integer 1 to those nodes of T at depth i represented by the first distinct tuple of the sorted list Q, assign the integer 2 to the nodes represented by the second distinct tuple, and so on.

Finding horizontal automorphisms. As with vertical automorphisms, fixed components of horizontal automorphisms play an important role.

Lemma 6. *Suppose that G is a series parallel digraph, where the children of the root in the canonical decomposition tree of G represent the components $G_1, G_2, \ldots, G_k$. Suppose that α is a horizontal automorphism of G.*
1. *If G is a parallel composition, then α fixes each one of $G_1, G_2, \ldots, G_k$.*
2. *If G is a series composition and k is even, then α has no fixed component.*
3. *If G is a series composition and k is odd, then α fixes $G_{(k+1)/2}$ and has no other fixed component.*

Proof. First suppose that G is a parallel composition. Suppose that G_1 is not fixed by α. The image of G_1 under α must be one of the components; suppose that it is G_2. Consider an upward planar drawing D which displays α, and a path p_1 in G_1 between s and t. The reflection of p_1 in the x axis is a path p_2 in G_2. It is clear that these paths must intersect (in D) on the x axis. This intersection cannot be at a vertex, since the only vertices that are shared between G_1 and G_2 are the source and the sink. Thus D has an edge crossing, contrary to hypothesis.

Now suppose that G is a series composition, and the components $G_1, G_2, \ldots, G_k$ are in order from source to sink. Note that G_1 is a leaf of the block-cutvertex tree of G, and so α must map G_1 to a leaf of the block-cutvertex tree. However, G_1 and G_k are the only leaves of the block-cutvertex tree. Since α is a horizontal automorphism, it swaps the source and sink of G, and it follows that G_1 maps to G_k. We can continue the argument to show that G_i maps to G_{k-i+1} for $1 \leqslant i < \lceil k/2 \rceil$. The second and third parts of the lemma follow. □

This lemma leads to the following theorem, which is the basis for our algorithm for finding horizontal automorphisms.

Theorem 3. *Suppose that G is a series parallel digraph.*
1. *If G is a parallel composition of $G_1, G_2, \ldots, G_k$, then G has a horizontal automorphism if and only if all of $G_1, G_2, \ldots, G_k$ have horizontal automorphisms.*
2. *Suppose that G is a series composition of $G_1, G_2, \ldots, G_k$. Let r be the root node of the canonical decomposition tree T of G.*
   (a) *If $tuple_H(r)$ is not a palindrome, then G has no horizontal automorphism.*
   (b) *If $tuple_H(r)$ is a palindrome and has even length, then G has a horizontal automorphism.*
   (c) *If $tuple_H(r)$ is a palindrome but has odd length, then G has horizontal automorphism if and only if the component of the "middle" node of the palindrome has a horizontal automorphism.*

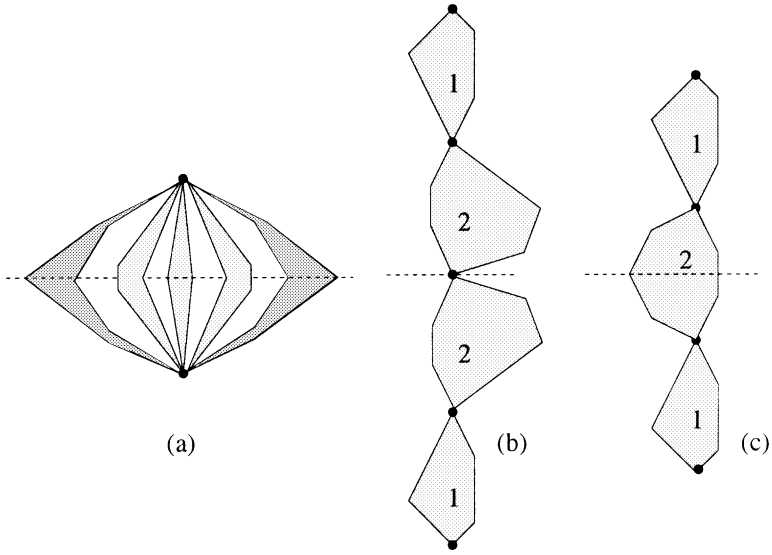

Fig. 9. Horizontal arrangements.

Proof. We deal with the parallel composition first. Suppose that G has a horizontal automorphism α. From Lemma 6, α fixes each one of $G_1, G_2, \ldots, G_k$. Thus any drawing which displays α also displays a horizontal automorphism of each of the components. Conversely, suppose that each of $G_1, G_2, \ldots, G_k$ has a horizontal automorphism. Then each of $G_1, G_2, \ldots, G_k$ has a drawing which displays an automorphism which swaps the source and sink. This drawing can be transformed using the croissant transformation above. The transformation preserves horizontal automorphism. Arranging the transformed drawings of $G_1, G_2, \ldots, G_k$ as in Fig. 9(a), we get a drawing of G which displays horizontal automorphism.

Now suppose that G is a series composition of $G_1, G_2, \ldots, G_k$. Let r be the root node of the canonical decomposition tree T of G. Note that in any upward drawing of G, G_i appears above G_j if and only if $i > j$.

If $tuple_H(r)$ is not a palindrome, there is more than one component that cannot be mapped to the component of the same isomorphism code. Thus we cannot construct a drawing of G which displays a horizontal automorphism. This means that G has no horizontal automorphism.

If $tuple_H(r)$ is a palindrome, then G_i is isomorphic to G_{k-i+1} for $1 \leqslant i < \lceil k/2 \rceil$. If k is even then we can construct a drawing of G which displays a horizontal automorphism by arranging $G_1, G_2, \ldots, G_k$ vertically, using the same drawing for G_{k-i+1} as for G_i, and identifying the sink of G_i with the source of G_{i+1} for each $i = 1, 2, \ldots, k-1$. This is illustrated in Fig. 9(b). Thus G has a horizontal automorphism.

Suppose that $tuple_H(r)$ is a palindrome and k is odd. If G has a horizontal automorphism, then from Lemma 6, the central component of G is fixed and thus must have a horizontal automorphism. Conversely, suppose that the component $G_{(k+1)/2}$ has a horizontal automorphism. Then $G_{(k+1)/2}$ has a drawing that displays a horizontal automorphism. We can construct a drawing of G that displays a horizontal automorphism in the same way as when k is even, except that $G_{(k+1)/2}$ is drawn in the middle; see Fig. 9(c). □

The overall horizontal algorithm can be described recursively as follows.

`horizontal_check`(r)
If r is a P node
then
(a) if `horizontal_check`(v) for every child v of r,
    then return(*true*)
    else return(*false*)
else if r is an S node
then
(a) If $tuple_H(r)$ is not a palindrome then return(*false*).
(b) If $tuple_H(r)$ is a palindrome and has even length then return(*true*).
(c) If $tuple_H(r)$ is a palindrome but has odd length
    then return(`horizontal_check`(v)), where v is the "middle" node of the palindrome.

Lemma 7. *Algorithms* `horizontal_labeling`, `horizontal_reverse` *and* `horizontal_check` *run in linear time and space.*

Proof. Using the same argument as for Lemma 3, we can show that algorithm `horizontal_labeling` runs in linear time and space; algorithms `horizontal_reverse` and `horizontal_check` are trivially linear. □

3.3. Rotational automorphisms

A rotational automorphism, like a horizontal automorphism, is direction-reversing, and we need a similar "reversing operation". This is followed by the labeling step, and finally a step to detect rotational automorphism. These steps are detailed below.

The reversing operation for rotational automorphisms. To detect rotational automorphism, we also need the reversing operation as a preprocessing step. This operation is very similar to the case for horizontal automorphism; the difference is that we cannot reverse the children of a P node, since we do not know a *priori* which children of a P node will be reversed by a rotational automorphism. In this case, the algorithm `rotational_reverse` is called after matching components in `rotational_check` below.

`rotational_reverse`(r)
If r is an S node
then
(a) Let $v_1, v_2, \ldots, v_k$ be the children of r, in the order of their composition, and suppose that $m = \lceil k/2 \rceil$.
(b) Mark each S node in the subtrees of the canonical decomposition tree T rooted at the nodes v_{m+1}, $\ldots, v_k$ as "reversed".

Rotational labeling. An algorithm `rotational_labeling` can be constructed in the same way as the algorithm `horizontal_labeling` for horizontal automorphisms. It computes two labels $tuple_R(u)$ and $code_R(u)$; we will omit details.

Finding rotational automorphisms. The algorithm for detecting rotational automorphisms has two parts, one dealing with parallel compositions and one dealing with series compositions. For series

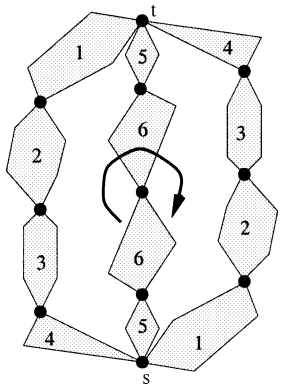

Fig. 10. Tuples for rotational automorphisms in a parallel composition.

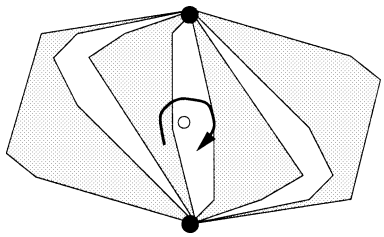

Fig. 11. Rotational arrangement.

compositions the algorithm is completely analogous to the algorithm for horizontal automorphisms. However, for parallel compositions it is a little more complex. The complexity arises because, in a rotational automorphism ρ, the image of a component of a parallel composition is "reversed" by ρ. This means that we must match the tuple of a component with its reverse. For example, in Fig. 10, the left component with $tuple_R = (4, 3, 2, 1)$ matches the right component with $tuple_R = (1, 2, 3, 4)$.

The algorithm is based on the following theorem.

Theorem 4. *Suppose that G is a series parallel digraph.*

1. *Suppose that G is a parallel composition of $G_1, G_2, \ldots, G_k$; suppose that the corresponding nodes of the canonical decomposition tree are $v_1, v_2, \ldots, v_k$. Form pairs (v_i, v_j), $1 \leqslant i < j \leqslant k$, such that $tuple_R(v_i)$ is the reverse of $tuple_R(v_j)$. Continue this pairing until no further pairs can be formed.*
   (a) *If there is more than one component that cannot be paired, then G has no rotational automorphism.*
   (b) *If all of $G_1, G_2, \ldots, G_k$ are paired, then G has a rotational automorphism.*
   (c) *If one component is not paired, then G has a rotational automorphism if and only if the unpaired component has a rotational automorphism.*
2. *Suppose that G is a series composition of $G_1, G_2, \ldots, G_k$. Let r be the root node of the canonical decomposition tree T of G.*
   (a) *If $tuple_R(r)$ is not a palindrome, then G has no rotational automorphism.*
   (b) *If $tuple_R(r)$ is a palindrome and has even length, then G has a rotational automorphism.*
   (c) *If $tuple_R(r)$ is a palindrome but has odd length, then G has a rotational automorphism if and only if the component of the "middle" node of the palindrome has a rotational automorphism.*

Proof. The proof of this theorem is similar to that of Theorem 3 and so we will only give an outline.

Firstly consider the case where G is a parallel composition of $G_1, G_2, \ldots, G_k$.

One can show, using methods such as in Lemma 5, that a rotational automorphism fixes at most one of the G_i. This implies that if there is more than one component that cannot be paired, then there is no way to construct a drawing of G which displays a rotational automorphism.

Now suppose that all $G_1, G_2, \ldots, G_k$ are paired. We can construct a drawing of G which displays a rotational automorphism by placing each $G_1, G_2, \ldots, G_k$ in an order such that each can be mapped with its reverse, and using the croissant transformation. This is illustrated in Fig. 11. It follows that G has a rotational automorphism.

Now suppose that one component G_i is not paired. Suppose that G has a rotational automorphism. One can show that G_i is fixed by the automorphism, and a drawing of G with rotational symmetry displays a rotational automorphism of G_i. Conversely, suppose that the component G_i has a rotational automorphism; it is an easy exercise, again using a croissant transformation, to construct a drawing of G which displays a rotational automorphism.

For a series composition, the proof is completely analogous to the proof of Theorem 3. □

The overall algorithm can be described recursively as follows.

`rotational_check`(r)
If r is a P node
then
(a) Pair the children $v_1, v_2, \ldots, v_k$ of r, so that for each pair (v_i, v_j), $tuple_R(v_i)$ is the reverse of $tuple_R(v_j)$.
(b) If more than one component is not paired then return(*false*).
(c) If all $G_1, G_2, \ldots, G_k$ are paired together then return(*true*).
(d) If one component is not paired
    then
      (i) let v is unpaired node
     (ii) `rotational_reverse`(v)
    (iii) `rotational_labeling`(T_v)
     (iv) return(`rotational_check`(v))
else if r is an S node
then
(a) If $tuple_R(r)$ is not a palindrome then return(*false*).
(b) If $tuple_R(r)$ is a palindrome and has even length then return(*true*).
(c) If $tuple_R(r)$ is a palindrome but has odd length then return(`rotational_check`(v)), where v is the "middle" node of the palindrome.
Note that in step (d), `rotational_labeling`(T_v) is to apply `rotational_labeling` to the subtree of T under v.

Lemma 8. *Algorithms* `rotational_labeling`, `rotational_reverse` *and* `rotational_check` *run in linear time and space.*

Proof. Using the same argument as for Lemma 3, we can show that algorithm `rotational_labeling` runs in linear time and space. Algorithm `rotational_reverse` is trivially linear.

For algorithm `rotational_check`, we need to show how to implement the pairing in linear time. For this, we construct the reverse $tuple'_R(v_i)$ of $tuple_R(v_i)$ for each i. Then we sort the set

$$\{tuple_R(v_1), tuple_R(v_2), \ldots, tuple_R(v_k), tuple'_R(v_1), tuple'_R(v_2), \ldots, tuple'_R(v_k)\}$$

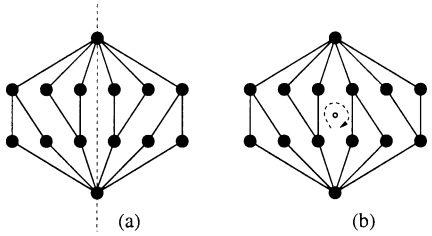

Fig. 12. Two drawings of a series parallel digraph.

lexicographically, and assign the integer 1 to every tuple that is least lexicographically, assign the integer 2 to every tuple that is second-least lexicographically, and so on. This gives a pair (c_i, c'_i) of integers to each node v_i. Two nodes v_i and v_j can then be paired if $c_i = c'_j$ and $c_j = c'_i$. □

3.4. Computing the maximum upward planar automorphism group

In this section, we describe the method for computing a maximum sized upward planar automorphism group. From Lemma 1, such a group is either

- trivial, or
- $\{1, p\}$ where p is either vertical, horizontal, or a rotation of order 2, or
- $\{1, p, q, r\}$, where the non-identity elements are one of each of the types vertical, horizontal, or rotation of order 2.

If one of the algorithms `vertical_check, horizontal_check` or `rotational_check` returns true, then the graph has a group of size 2 or more. We noted at the start of this section, using Figs. 4(b) and 4(c), that the existence of a rotational automorphism and a horizontal automorphism is not sufficient to ensure that the group has size 4. Further, Fig. 12 shows two drawings of a series parallel digraph, which show that the existence of a rotational automorphism and a vertical automorphism is not sufficient to ensure that the group has size 4.

However, we can show that the existence of a horizontal and a vertical automorphism ensures that the group has size 4.

Theorem 5. *Suppose that G is a series parallel digraph with a maximum size upward planar automorphism group of size m. Then*:

1. *If G has no vertical, horizontal or rotational automorphisms, then $m = 1$.*
2. *If G has only one automorphism among vertical, horizontal or rotational, then $m = 2$.*
3. *If G has both horizontal and rotational automorphisms but no vertical automorphism, then $m = 2$.*
4. *If G has both vertical and rotational automorphisms but no horizontal automorphism, then $m = 2$.*
5. *If G has a vertical and a horizontal automorphism, then $m = 4$ and the graph has a maximum of the form $\{1, p, q, r\}$, where the non-identity elements are one of each of the types vertical, horizontal and rotation of order 2.*

Proof. The first four parts are trivial; we consider only part (5). Suppose that G has a vertical and a horizontal automorphism, G is a composition of $G_1, G_2, \ldots, G_k$. Assume for the moment that k is even.

First, suppose that G is a parallel composition. From Theorem 3, we can construct drawings of each of $G_1, G_2, \ldots, G_k$, each of which displays a horizontal automorphism. Since G has a vertical

automorphism, one can deduce from Theorem 2 that the isomorphism classes of the G_i all have an even number of elements. By pairing these components and using a croissant transformation, one can construct a drawing which displays both the horizontal and the vertical automorphism. The product of these two symmetries is a rotational symmetry.

Now suppose that G is a series composition. From Theorem 2, one can deduce that each of $G_1, G_2, \ldots, G_k$ has a drawing with vertical symmetry. Further, from Theorem 3, the order of the components from the source to sink gives a palindrome in isomorphism codes. It is simple to construct a drawing with both vertical and horizontal symmetry; this has rotational symmetry.

Slight extensions to the arguments above cover the case where k is odd. □

The procedure to output the maximum upward planar automorphism group simply checks each case of Theorem 5.

4. Drawing algorithms

A topological embedding of a series parallel digraph is defined by the order of the P nodes in the canonical decomposition tree. It is simple to adjust the algorithms `vertical_check`, `horizontal_check` and `rotational_check` to sort the children of each P node to obtain an embedding which respects the automorphism.

In the next section, we show how to use this embedding to obtain a symmetric visibility drawing. Based on this drawing, other drawing algorithms, described in the following sections, may be derived.

4.1. Visibility drawings

First we describe a simple procedure for giving a visibility representation [16] of a series parallel digraph G. In the representation that we construct, the horizontal line segment for the source is a vertical translation of the horizontal line segment of the sink.

For a graph which consists of a single edge, such a representation is simple. Suppose that D_1 and D_2 are visibility representations of series parallel digraphs G_1 and G_2, respectively. If G is a series composition of G_1 and G_2, then we can construct a representation D of G by "stretching" the narrower of D_1 and D_2 and identifying the source of one with the sink of the other; see Fig. 13(a). If G is a parallel composition of G_1 and G_2, then we can construct a representation D of G by "stretching" the shorter of D_1 and D_2 and identifying their sources and sinks; see Fig. 13(b).

Two traversals of the canonical decomposition tree can be used to compute the visibility representation. One traversal computes the size of the enclosing rectangle for each component, the next computes the route for each edge. This works in linear time.

The algorithms `vertical_check, horizontal_check` and `rotational_check` give an ordering of the children of each parallel component in the decomposition tree. Our algorithm places parallel components from left to right across the page in the same order. One can easily deduce the following theorem.

Theorem 6. *There is a linear time algorithm which constructs a visibility drawing of a series parallel digraph such that a maximum size upward planar automorphism group is displayed.*

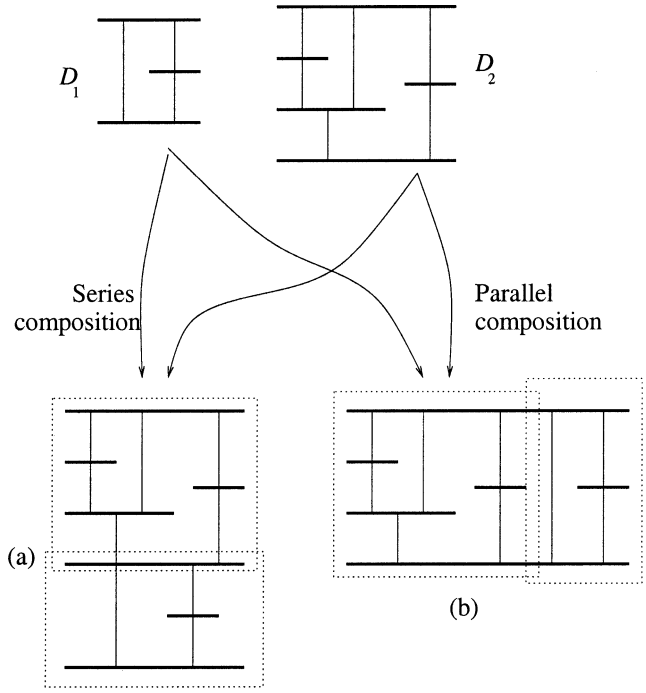

Fig. 13. Constructing visibility representations of series parallel digraphs.

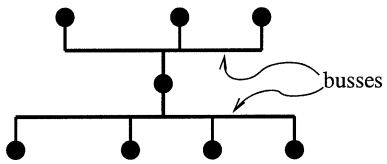

Fig. 14. The busses about a vertex.

4.2. Bus-orthogonal drawings

A simple transformation of the visibility representation gives a *bus-orthogonal* drawing. In the bus-orthogonal drawing, a vertex is connected to its neighbor via a *bus*. Each non-source vertex v has a bus, which is a horizontal line just below v. Similarly, each non-sink vertex v has a bus, which is a horizontal line just above v. In some cases (for example, if the in or out degree is one), the horizontal lines of the bus may have length zero. The neighbors of v are connected to the appropriate bus by vertical line segments. These concepts are illustrated in Fig. 14.

The transformation from a visibility representation to a bus-orthogonal drawing is illustrated in Fig. 15. First thicken each horizontal segment in the visibility representation, and place a vertex in the center of each rectangle. Then the busses are drawn along the bottoms and tops of these rectangles, as far as the vertical lines representing edges indicate. Then the vertices are joined to the busses, and the rectangles are removed.

Fig. 16(b) shows a bus-orthogonal drawing obtained from Fig. 16(a).

The following theorem summarizes.

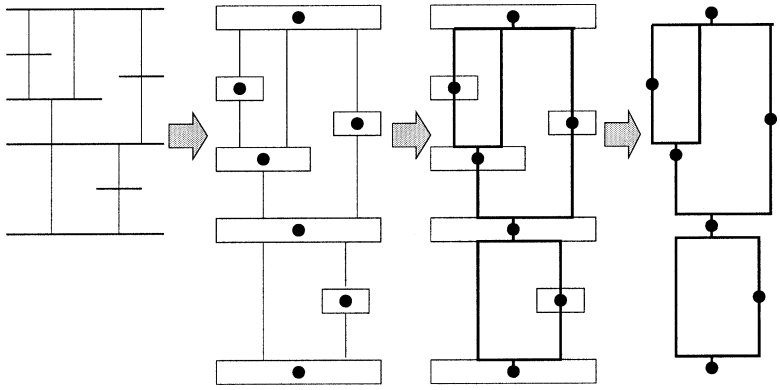

Fig. 15. Transformation to bus-orthogonal drawings.

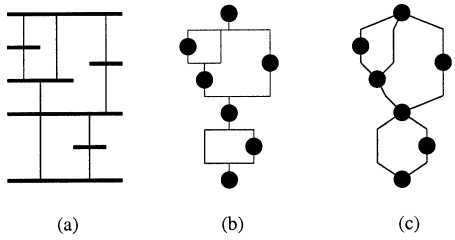

Fig. 16. Transformations to the bus-orthogonal drawing and the polyline drawing.

Theorem 7. *There is a linear time algorithm which constructs a bus-orthogonal drawing of a series parallel digraph such that the output is planar, and a maximum size upward planar automorphism group is displayed.*

4.3. Polyline drawings

One can use a standard transformation from visibility drawings to polyline drawings; for example, Fig. 16(c) is obtained from Fig. 16(a). This gives the following theorem.

Theorem 8. *There is a linear time algorithm which constructs an upward planar polyline drawing of a series parallel digraph such that each edge has at most two bends, and a maximum size upward planar automorphism group is displayed.*

5. Remarks

In this paper we have described an algorithm for computing the maximum size upward planar automorphism group of a series parallel digraph. These automorphisms may be used to construct drawings of series parallel digraphs with maximum symmetry. In this section we conclude with some remarks.

5.1. Vertex series parallel digraphs

A digraph whose line digraph is a series parallel digraph is a *minimal vertex series parallel digraph*, and a digraph whose transitive reduction is a minimal vertex series parallel digraph is a *vertex series parallel digraph*; see [17]. It would be interesting to extend the results of this paper to minimal vertex series parallel digraphs. However, extension to vertex series parallel digraphs is unlikely, because the isomorphism problem for this broader class of graphs is isomorphism complete.

5.2. Area

The drawings obtained in the algorithms of Section 4 are not grid drawings. However, they do have good area bounds. Consider the visibility drawing described in Section 4.1. The "stretching" operation has the following property. Suppose that the width and height of a drawing D are $width(D)$ and $height(D)$ respectively. If D_s is the visibility drawing resulting from the series composition of drawings D_1 and D_2, then

$$width(D_s) = \max\big(width(D_1), width(D_2)\big),$$
$$height(D_s) = height(D_1) + height(D_2).$$

Also, if D_p is the visibility drawing resulting from the parallel composition of drawings D_1 and D_2, then

$$width(D_p) = width(D_1) + width(D_2),$$
$$height(D_p) = \max\big(height(D_1), height(D_2)\big).$$

From these equations, assuming that a single edge is drawn with width and height one, we can deduce that the width and height of the visibility drawing of an n vertex series parallel digraph are both $O(n)$. Since the "stretching" operation never decreases distances, the drawing has a minimum distance of one between any pair of nodes. The bus orthogonal and polyline drawings are also $O(n) \times O(n)$, because they differ in width and height by a constant only.

Bertolazzi et al. [4] have shown that a straight-line drawing of a series parallel digraph, with a given embedding, may require exponential area. One can easily extend the result of Bertolazzi to show that a symmetric straight-line drawing may require exponential area.

5.3. Near-symmetric drawings

There are many possible formalizations of the intuitive notion of a drawing being "nearly symmetric". In some cases, it is possible to use the algorithms of the preceding sections to draw graphs in a "nearly symmetric" way. We list two such cases below. However, despite these cases, the precise modeling of "near symmetry" and construction of drawings displaying near symmetry remains a challenge.

Symmetric subgraphs. It is often the case that a graph has no nontrivial upward planar automorphism group but has components which have some non-trivial upward planar automorphisms. Our algorithm partially addresses this problem.

If there is an isomorphism between two components G_1 and G_2 of G, then a careful adaptation of the algorithms (for example, by prioritizing rotational automorphisms above horizontal and vertical

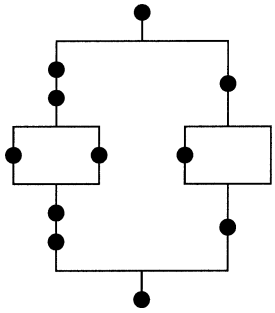

Fig. 17. A drawing displaying a kind of "near symmetry".

automorphisms), we can ensure that the drawings of G_1 and G_2 differ by a translation and at most one reflection.

However, we should note that there may be pairs of subgraphs (not components) which are isomorphic but are not drawn congruently. For example, "near symmetry" along the lines of [3] or [5] is not achieved.

Paths and edges. The middle drawing in Fig. 1 has the appearance of vertical symmetry. However, this "symmetry" maps paths of length 2 to single edges, and so the reflection in the y axis does not, strictly speaking, induce a vertical automorphism.

One can ensure that drawings display this kind of symmetry with the following trick. For the canonical decomposition tree, consider all S nodes u all of whose children are leaf nodes. Such a node represents a path in the graph. We can replace this path by a single edge, effectively deleting the children of u. The application of the algorithms in the preceding sections give a drawing in which one path may be mapped to another, regardless of the length of the path.

A drawing displaying this kind of "near symmetry" is in Fig. 17.

References

[1] A. Aho, J. Hopcroft, J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.
[2] L. Babai, Automorphism groups, isomorphism, and reconstruction, in: Graham, Groetschel, Lovasz (Eds.), Handbook of Combinatorics, Vol. 2, Elsevier, Amsterdam, 1995, Chapter 27.
[3] S. Bachl, Isomorphic subgraphs, graph drawing, in: J. Kratochvil (Ed.), Lecture Notes in Computer Science, Vol. 1731, Springer, Berlin, 1999, pp. 286–296.
[4] P. Bertolazzi, R.F. Cohen, G. di Battista, R. Tamassia, I.G. Tollis, How to draw a series–parallel digraph, Internat. J. Comput. Geom. Appl. 4 (4) (1994) 385–402.
[5] K.-W. Chin, H.-C. Yen, Algorithms and complexity analysis of the symmetry number for graphs, manuscript, 1998.
[6] R.F. Cohen, G. di Battista, R. Tamassia, I.G. Tollis, Dynamic graph drawing: trees, series–parallel digraphs, and planar st-digraphs, SIAM J. Comput. 24 (5) (1995) 970–1001.
[7] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice-Hall, Englewood Cliffs, NJ, 1998.
[8] P. Eades, X. Lin, Spring algorithms and symmetry, Theoret. Comput. Sci. 240 (2) (2000) 379–405.

[9] S.-H. Hong, P. Eades, S.-H. Lee, Finding planar geometric automorphisms of planar graphs, in: Chwa, Ibarra (Eds.), Algorithms and Computation, Lecture Notes in Computer Science, Vol. 1533, Springer, Berlin, 1998, pp. 277–286.
[10] X. Lin, Analysis of Algorithms for Drawing Graphs, Ph.D. Thesis, University of Queensland, 1992.
[11] A. Lubiw, Some NP-complete problems similar to graph isomorphism, SIAM J. Comput. 10 (1) (1981) 11–21.
[12] J. Manning, M.J. Atallah, Fast detection and display of symmetry in trees, Congr. Numer. 64 (1988) 159–169.
[13] J. Manning, M.J. Atallah, Fast detection and display of symmetry in outer-planar graphs, Discrete Appl. Math. 39 (1992) 13–35.
[14] J. Manning, Geometric Symmetry in Graphs, Ph.D. Thesis, Purdue University, 1990.
[15] R.A. Mathon, A note on graph isomorphism counting problem, Inform. Process. Lett. 8 (1979) 131–132.
[16] R. Tamassia, I.G. Tollis, A unified approach to visibility representations of planar graphs, Discrete Comput. Geom. 1 (1986) 321–341.
[17] J. Valdes, R. Tarjan, E. Lawler, The recognition of series–parallel digraphs, SIAM J. Comput. 11 (2) (1982) 298–313.
[18] J. Valdes, Parsing flowchart and series–parallel graphs, Technical Report STAN-CS-78-682, Computer Science Department, Stanford University, 1978.
[19] H. Wielandt, Finite Permutation Groups, Academic Press, New York, 1964.